# Technical Report

TR-UPM-GAPS-10-01

# Compiling for an ARM architecture: iMote2-TinyOS

Nelson I. Dopico

v1.0, July, 2010

## Index



---

## Abstract

In this technical report you can find pointers to user-contributed compiler packages both in elf and none-eabi mode for the ARM architecture. Instructions for compiling as well as summarized technical information to help understand the reasoning involved are given so that users know concisely how to configure their TinyOS environment and to facilitate other ARM programmers' work. It mainly focuses on the tandem iMote2-TinyOS, but other users may find it useful either because of the software links or the information regarding instruction sets, float point operations or compiling options.

## Acknowledgements

The author thanks Carlos Gil and Iñigo Arrazola for their feedback on testing the compilers. The work has been sponsored by the Community of Madrid, the European Social Fund and the project COMONSENS (CSD2008-00010 COMONSENS) funded by the Ministry of Science and Innovation of Spain.

# 1 Introduction

Programming for certain embedded systems can turn out to be a nightmare not only for the sake of coding and debugging, but because of missing tools (compilers) or technical information. If everything is wrapped up with an embedded operating system such as TinyOS, the goal becomes even tougher. To keep fresh our memory and to help others who could face similar issues, this technical report provides an overview on how to compile for the ARM architecture focusing on the XScale family and more precisely on the PXA271 as it is the processor which iMote2 integrates. The approach aims primarily at helping TinyOS users with the instructions and reasoning involved to adjust a regular TinyOS installation for our objective are detailed; however, other embedded system programmers can take advantage of the enclosed information or the referred software packages.

PXA271 is a 32-bit processor which was initially manufactured by Intel, but was resold to Marvell along with the XScale processor family [1]. Nowadays, Intel does not provide any development environment or compiler except Wasabi, which has not been updated since 2003. Marvell provides under non disclosure agreement a cross compiler for Windows environments.

The report is structured into 6 more sections. Section 2 explains the instruction sets that can be executed on PXA271. Section 3 focuses on floating point operations in the ARM architecture and 27X series in particular. Based on the aforementioned section, Section 4 explains the compilation options that seem to fit better PXA271 for each compiling mode. Section 5 deals with the compiler installation. Section 6 specifies how to edit TinyOS config files to get the compilers up and running with the desired compilation options. Conclusion (Section 7) comprises a quick guide to tune TinyOS with the compiler and the suggested options.

# 2 Instruction set

XScale family implements the instruction set ARMv5, but PXA27X series (inclusive) and superior integrate a co-processor (or iwMMXt) which implements additional instructions (MMX) and are able to process differently float numbers. As a result, the PXA271 instruction set is a super set of ARMv5 which comprises both ARMv5te (specific set for XScale) and iwMMXt. Compilation options usually refer to the whole set as iwmmxt.

# 3 Floating point operations

Two acronyms are used related to this kind of operations:

- FPA. Floating Point Accelerator
- VPU. Vector Floating Point

Each of the aforementioned is one of a kind of floating point operations. FPA is the oldest and most classical and is implemented by default in files ABI v0 – Old-ABI. VPU is the latest and it is just implemented by certain cores, namely ARM 9/10/11 [2]. XScale cores are not the same across the whole family, but are defined by the processor itself: this report regards PXA271. For practical purposes we will just focus on the fact that the XScale architecture does not implement VPU, but is able to operate in FPA mode and it is the default mode of executable files generated by elf compilers – though such mode is required to be explicitly indicated as a parameter at times.

Nevertheless, 27X series differ from earlier ones as 27X supports the iwMMXt instruction set along with sixteen new 64-bit data registers and eight 32-bit control registers as well as machine code instructions (operation codes) -which overlap with the FPA floating point extension- and the capability

to operate integers in SIMD mode [3,1]. As a consequence, if PXA271 operates with the complete iwMMXt instruction set, it will not use FPA as these two are mutually incompatible. iwMMXt has its own way to operate float numbers and according to the information available it seems to perform better than FPA, but it is overcome by VPU [2,4].

Upon compiling for PXA271, depending on the compiler capabilities, one has to choose between compiling for the XScale architecture in FPA mode or for the iwMMXt architecture. Our microprocessor supports both, but the latter is recommended as it is theoretically better – see Section 2.

# 4 Compiling Options

Useful compiling options for the PXA27X architecture are:

-mcpu. This specifies the name of the target ARM processor. GCC uses this name to determine what kind of instructions it can emit when generating assembly code.

-mfpu. This specifies what floating point hardware (or hardware emulation) is available on the target.

-mtune. This option is very similar to the -mcpu= option, except that instead of specifying the actual target processor type, and hence restricting which instructions can be used, it specifies that GCC should tune the performance of the code as if the target were of the type specified in this option, but still choosing the instructions that it will generate based on the CPU specified by a -mcpu= option. For some ARM implementations better performance can be obtained by using this option.

-march. This specifies the name of the target ARM architecture. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. This option can be used in conjunction with or instead of the -mcpu= option.

| Option | Accepted values |
|---|---|
| mcpu | `xscale`<br>`iwmmxt` |
| mtune | |
| mfpu | `fpa` |
| march | `armv5te`<br>`iwmmxt`<br>`iwmmxt2` * |

Next two sections provide information on compiling options for two different compiling modes: elf and none-eabi. ELF stands for Executable and Linkable Format while EABI means Embedded-Application Binary Interface. The former is the most classical output file version, i.e. for obj and exec files as well as shared libraries and core dumps [5]. The later is more modern and specifies standard conventions for file formats, data types, register usage, stack frame organization, and function parameter passing of an embedded software program [6]. In practice, EABI seems to perform better regarding floating point operations [4]. The prefix none in none-eabi refers to the fact that the compiler has not been built for a specific operating system, as it is necessary for TinyOS.

* The option iwmmxt2 is just available from 4.4.1 version on and no information was found as of July, 2010 whether there is support or not by PXA27X series.

## 4.1 GCC elf

The latest elf package available on our website at the time that this report was written (arm-compiler-elf_4.1.1-1.deb) was v4.1.1 and instructions are based on tests with it, however, newer versions could be uploaded *a posteriori* though the same behavior is expected with updates. Such package includes gcc-4.1.1, binutils-2.17 and newlib-1.14.

Experience says that it is compulsory to state -mfpu=fpa so that TinyOS (as of v2.1) is able to generate main.exe, otherwise compatibility errors will arise between the TinyOS object code and default compiler-owned libraries. If just the option -mfpu=fpa is used, the output file can be executed by a PXA271, though the exec code is a generic version for the ARM family. In order to raise the performance, an option to specify the xscale instruction set to be used can be included (`-mcpu=xscale` with `-mtune=xscale` and `-mfpu=fpa`). Although it will not use fully PXA271 characteristics, it seems to be the best option found so far for the elf-compiler version – old ABI or ABI v0.

Preferred options → `-mcpu=xscale -mtune=xscale -mfpu=fpa`

## 4.2 GCC none-eabi

The latest none-eabi package available on our website at the time that this report was written (arm-compiler-none-eabi-4.3.3-3.deb) was v4.3.3 and instructions are based on tests with it; however, newer versions could be uploaded *a posteriori* though the same options are expected to be accepted by updates. Such package includes gcc-4.3.3, binutils-2.19.1 and newlib-1.17.

This compiler allows us to use the same combination as in the previous one (`-mcpu=xscale` with `-mtune=xscale` and `-mfpu=fpa`) and other suboptimal ones (`-mcpu=xscale`, `-mcpu=iwmmxt`, or `-mcpu=iwmmxt` with `-mtune=iwmmxt`) – in any case we will obtain an EABI-v4 exec file. Notice that it is possible to compile in none-eabi mode and still build an output file with FPA, though it will not perform the best.

For this compiler version it would be advisable to use `-mtune=iwmmxt` with `-march=iwmmxt`. The former asks the compiler to tune the high-level code performance as if the target architecture/processor were the one stated after the equal symbol. It does not imply that the instruction set is specific for such architecture – theoretically it would be valid for any ARM unless extra parameters are stated. The latter option (`-march=iwmmxt`) indicates the compiler to generate assembly code specifically for a given architecture.

Preferred options → `-mtune=iwmmxt -march=iwmmxt`

# 5 Compiler Installation

If you decide to install the deb packages provided at www.gaps.ssr.upm.es/en/research/wsn you just need to download your preferred compiler version and use Synaptic, GDebi Package Installer, apt-get or another overlay application. In most Gnome environments you will just need to double click on the file and GDebi will run.

Compilers provided are installed under `/usr/arm/elf` for elf versions and `/usr/arm/none-eabi` for none-eabi. Such decision aims at avoiding incompatibility issues with other cross compilers as the whole chain is self-contained within the same directory tree. Besides the aforementioned directory tree, three files (soft links) are created to ease the compiler usage in the `/usr/local/bin` directory: arm-<version>-objdump , arm-<version>-objcopy and arm-<version>-gcc where <version> can be elf or none-eabi. These soft links are there so that users can call them from the command line by

simply typing their names.

If you ever wish to check out man pages regarding this installation you will need to move to the specific directory or type the full path: `/usr/arm/<version>/man/man1/<man_file>`.

Compiler uninstallation can be performed by any package manager interface the same way you would do with other packages.

# 6 Editing TinyOS config files for compilation

Supposed that you are using TinyOS, two files should be edited to set up proper compilation options and calls as detailed in previous sections.

**/opt/tinyos-2.1/support/make/pxa27x.rules**. In the event you are using another version than TinyOS 2.1, be aware that changes are:

- Updating names for gcc, objcopy and objdump exec files.
- Adding compilation flags as it is already detailed along with a minor change to ensure that all the source files (assembly and c) are passed to the compiler at once, otherwise the compiler will report an error.

Below you have the lines that need to be substituted. Commented ones are meant for usage with elf versions and uncommented for none-eabi.

```
#GAS = arm-elf-gcc -combine -c # This ensures .c and .s
compiled object are compatible
#OBJCOPY = arm-elf-objcopy
#OBJDUMP = arm-elf-objdump
#PFLAGS += -mcpu=xscale -mtune=xscale -mfpu=fpa

GAS = arm-none-eabi-gcc -combine -c # This ensures .c
and .s compiled object are compatible
OBJCOPY = arm-none-eabi-objcopy
OBJDUMP = arm-none-eabi-objdump
PFLAGS += -mtune=iwmmxt -march=iwmmxt
```

**/opt/tinyos-2.1/tos/platforms/intelmote2/.platform**. Here, just an update on the gcc actual name is required. If elf version is used, then it should be arm-elf-gcc.

```
  -gcc=arm-none-eabi-gcc
```

# 7 Conclusion

Various sections in the current report intend to provide the reader with specific knowledge to understand how her/his hardware (namely iMote2, PXA271 or another XScale processor) operates, its characteristics and how to set up different compilation options for performance optimization.

If the reader is just interested in the outcomes rather than in the explanations, then what is required to

be done is:

a) Select a compiler version: elf or none-eabi and download it from www.gaps.ssr.upm.es/en/research/wsn/138-compiling-for-an-arm-architecture-imote2-tinyos

b) Install the package. Double clicking will be enough to start a package manager for most Linux distributions supporting deb packages.

c) Edit two configuration files in the TinyOS tree (support/make/pxa27x.rules and tos/platforms/intelmote2/.platform). Edition details are squared in Section 6.

In the event that you find useful this information and use any part of our work (either the information enclosed in this technical report or just the packages themselves) we would be grateful if you cite us:

**IEEE Style**:

N. Dopico, "Compiling for an arm architecture: imote2-tinyos," Universidad Politecnica de Madrid (UPM), Madrid, Spain, Tech. Rep. TR-UPM-GAPS-10-01, July 2010.

**ACM Style**:

N. Dopico. Compiling for an arm architecture: imote2-tinyos. Technical Report TR-UPM-GAPS-10-01, Universidad Politecnica de Madrid (UPM), Madrid, Spain, July 2010.

**Bibtex**:

```
@TECHREPORT{dopico10_iMote,
  author = {{Nelson I.} Dopico},
  title = {Compiling for an ARM architecture: iMote2-TinyOS},
  institution = {Universidad Politecnica de Madrid (UPM)},
  year = {2010},
  number = {TR-UPM-GAPS-10-01},
  address = {Madrid, Spain},
  month = {July}
}
```

# References


[1] "XScale," *Wikipedia, the free encyclopedia*, Jun. 2010.

[2] "ArmEabiPort," *Debian Wiki*, Mar. 2010.

[3] "MMX (instruction set)," *Wikipedia, the free encyclopedia*, Jul. 2010.

[4] A. Calderon and N. Castillo, "Why ARM's EABI matters," *Linux for Devices*, Mar. 2007.

[5] "Executable and Linkable Format," *Wikipedia, the free encyclopedia*, Jun. 2010.

[6] "Application binary interface," *Wikipedia, the free encyclopedia*, Apr. 2010.